# Computer Organization

# Multi-cycle CPU

Gyuhyeon Lee (guhylee@snu.ac.kr)
High Performance Computer System (HPCS) Lab

April 13, 2017

## Objective

- To implement a multi-cycle CPU, which supports full TSC ISA.

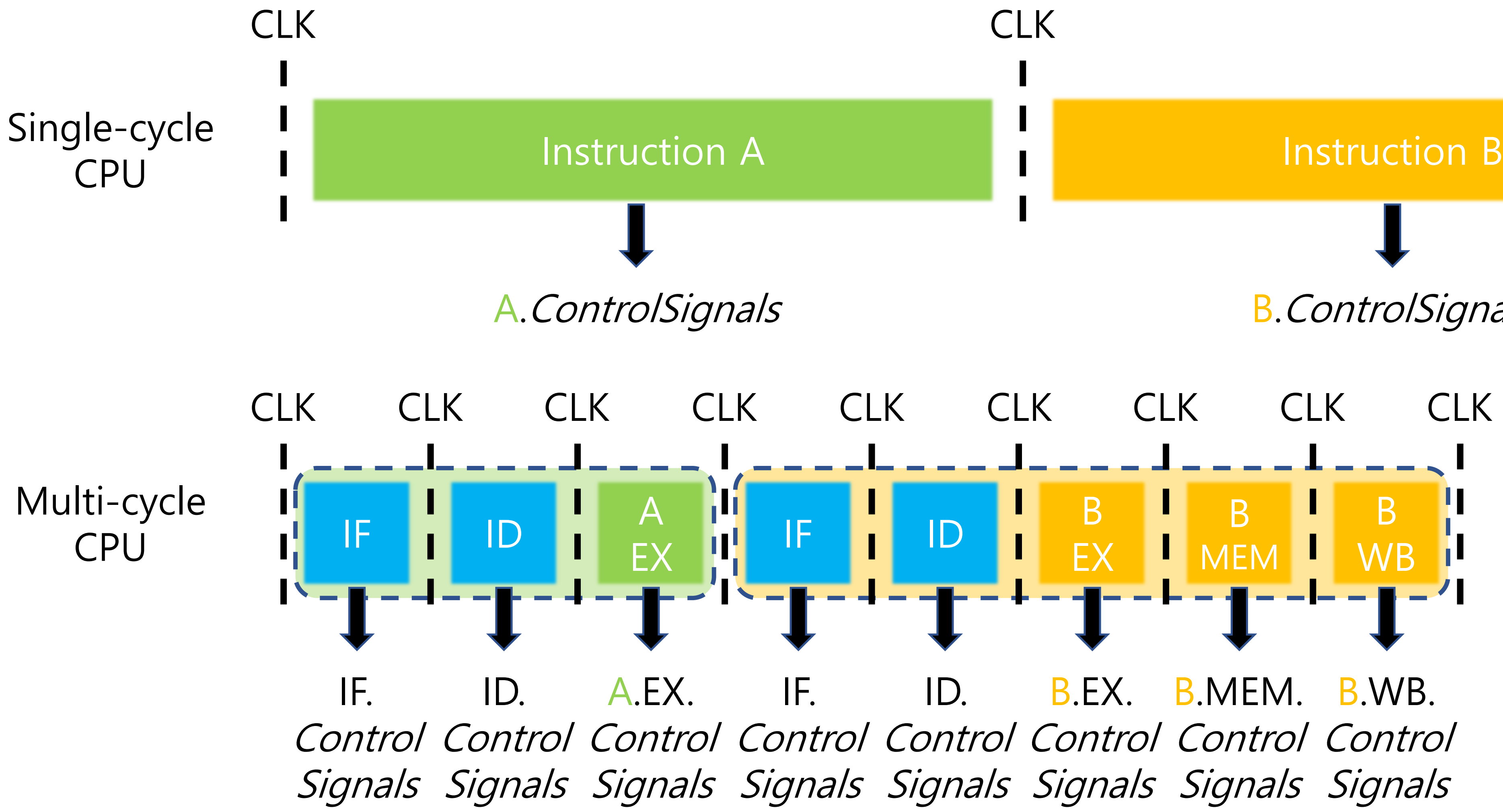
## Details of Objective

- To design a datapath & a control unit of the multi-cycle CPU
- You are free to revise your previous (single-cycle) CPU,
  or design it from scratch.

## Full TSC supports

- Please refer to the **[Assn4] tsc_ISA.rev.1.pdf** file.
    - The previous one has lots of typos, so I revised it.

# Control unit of Multi-cycle CPU

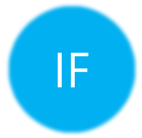


A is a simpler instruction which requires only IF/ID/EX stages.

B is more complex and requires every stage.

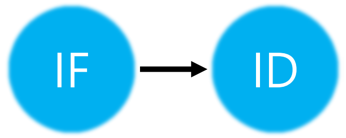
e.g.,
A : Branch, B : Load

## Control unit of Multi-cycle CPU



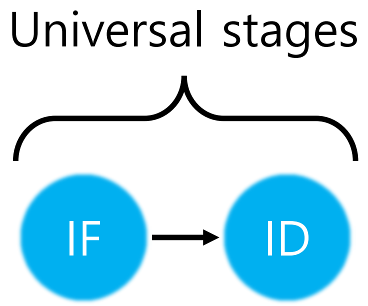
The execution starts from the IF stage.

## Control unit of Multi-cycle CPU



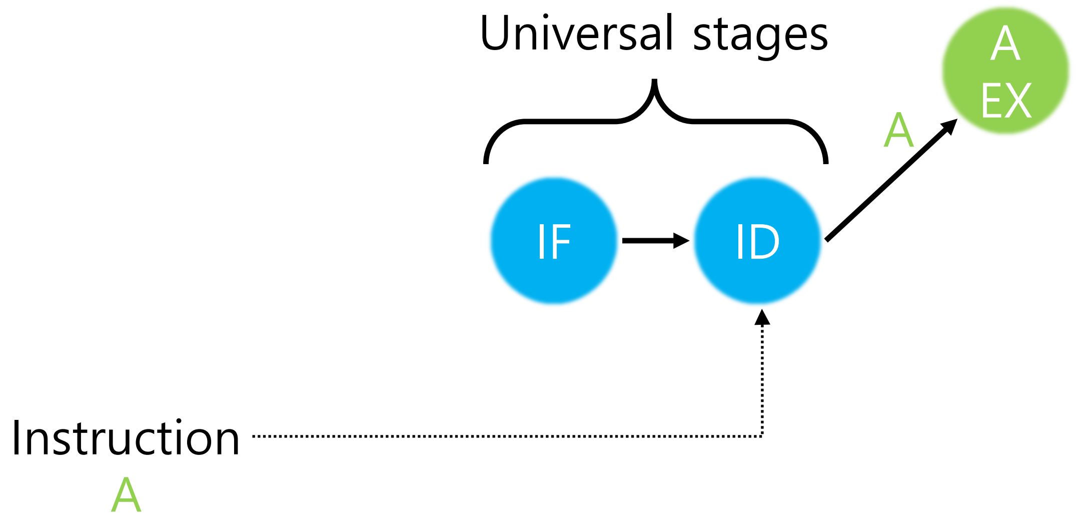
Then, it shifts to the ID stage.

# Control unit of Multi-cycle CPU



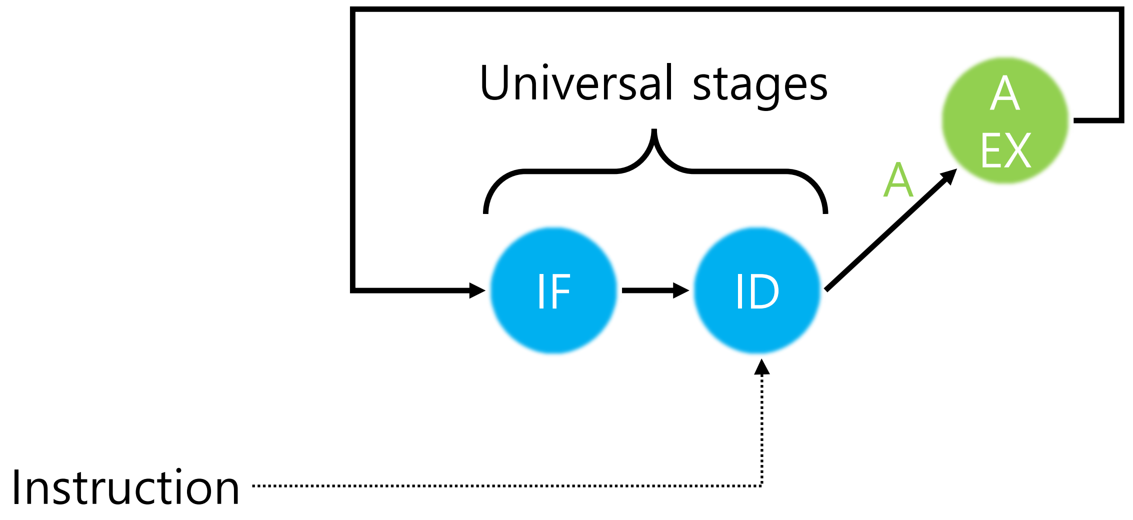
These two stages are universal,
because it doesn't know the type of instruction yet.

# Control unit of Multi-cycle CPU



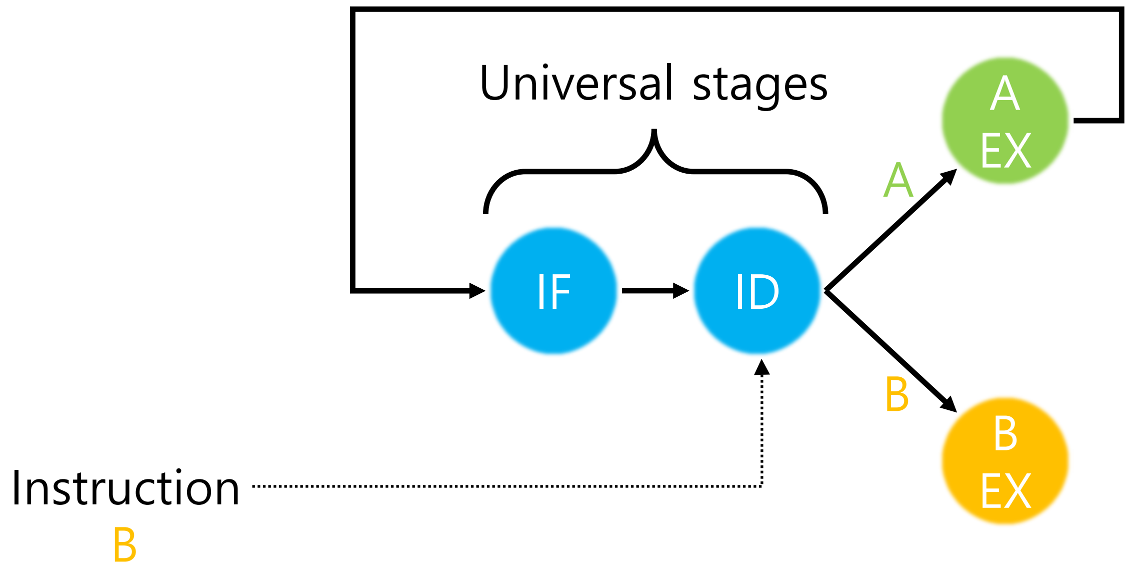
After decoding, it realizes the instruction type.
Let's assume that the instruction is A.

# Control unit of Multi-cycle CPU



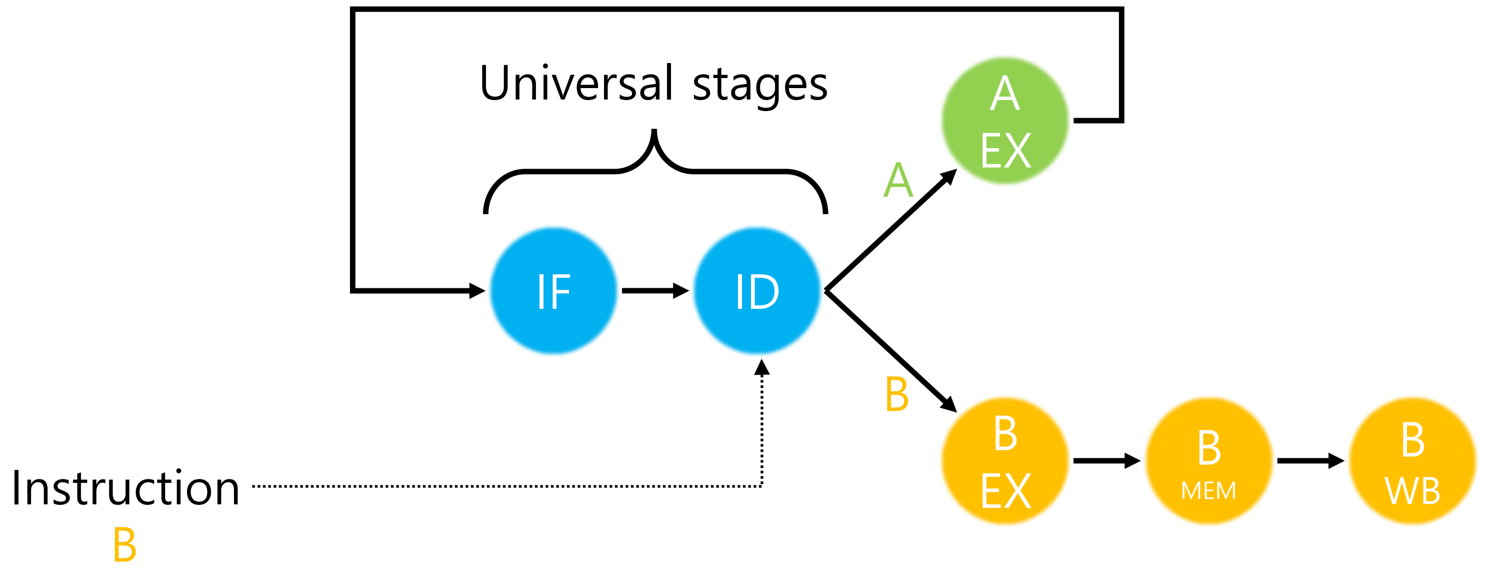
After it finishes execution, it has to return to the IF stage.

# Control unit of Multi-cycle CPU



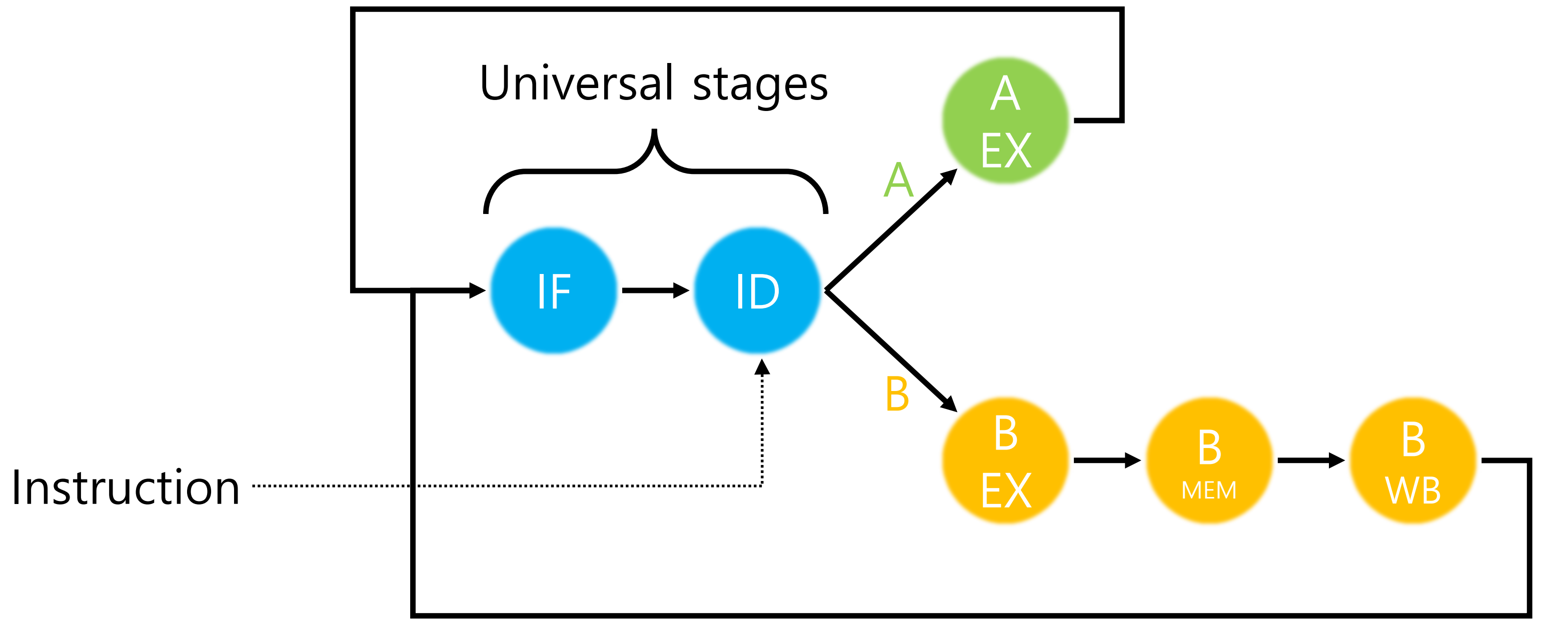
Let's assume that the next instruction is B

# Control unit of Multi-cycle CPU



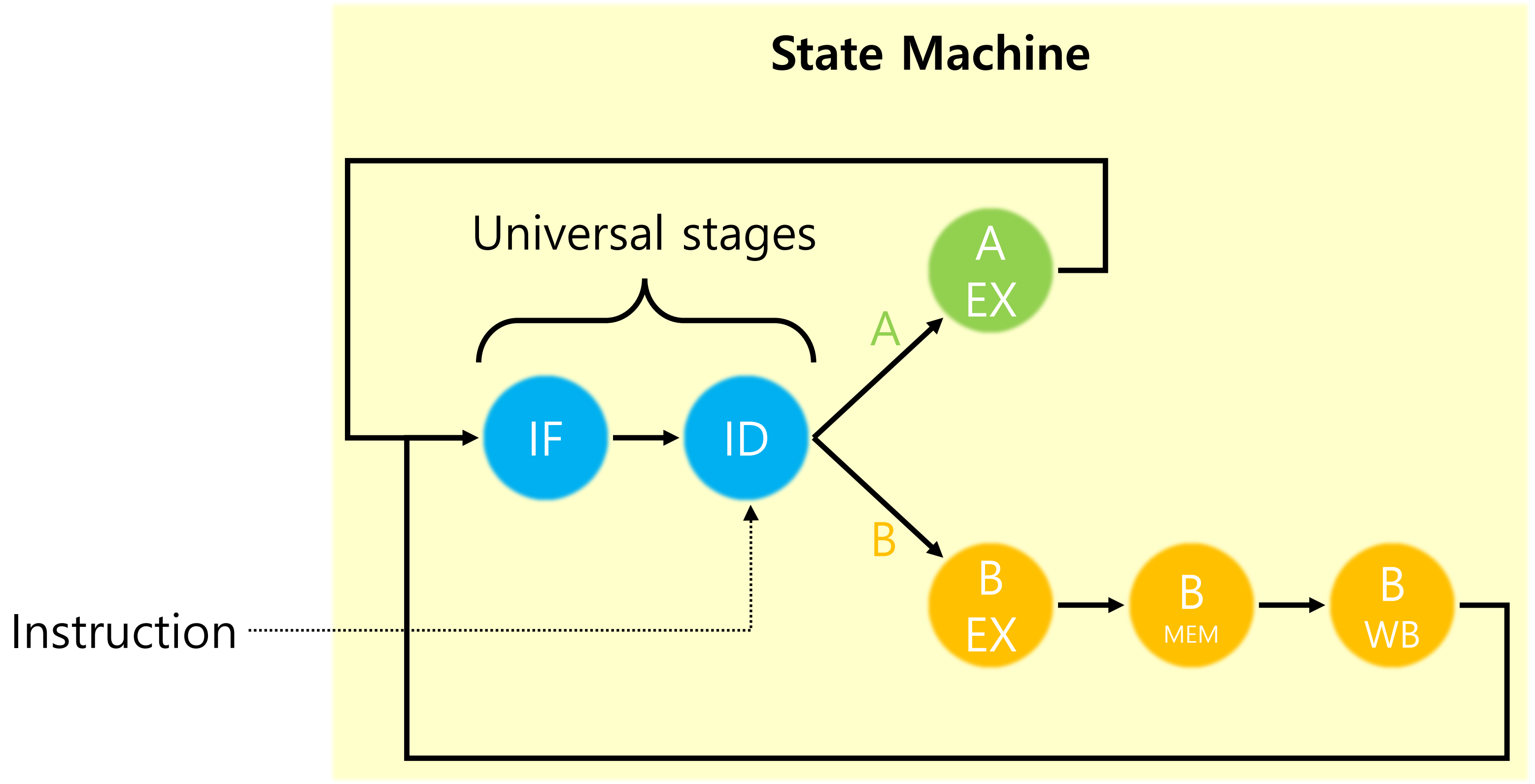
The instruction B requires the MEM and WB stage.

# Control unit of Multi-cycle CPU



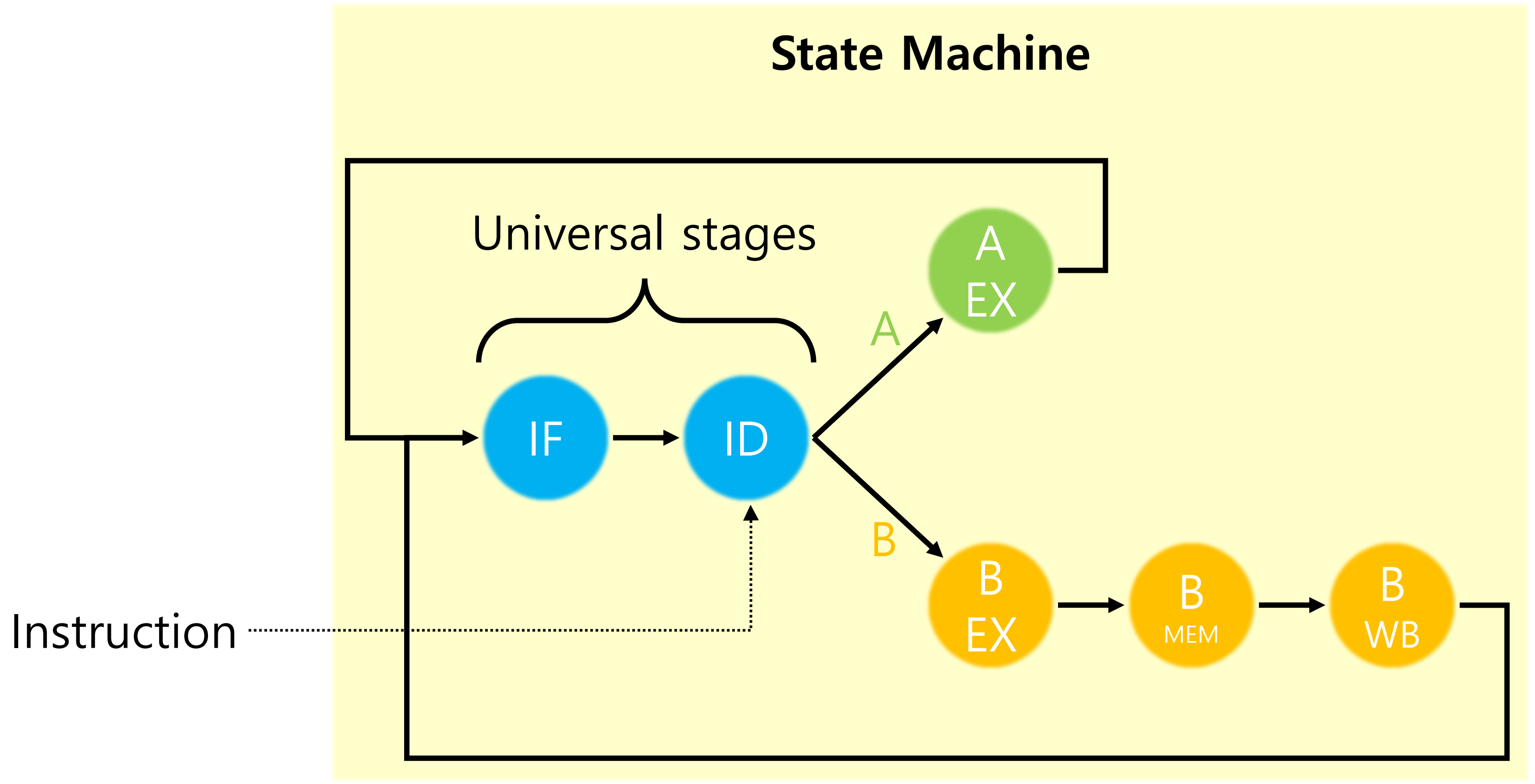
After it finishes execution, it also has to return to the IF stage.

# Control unit of Multi-cycle CPU



All of these stages form the state machine for the control unit.

# Control unit of Multi-cycle CPU



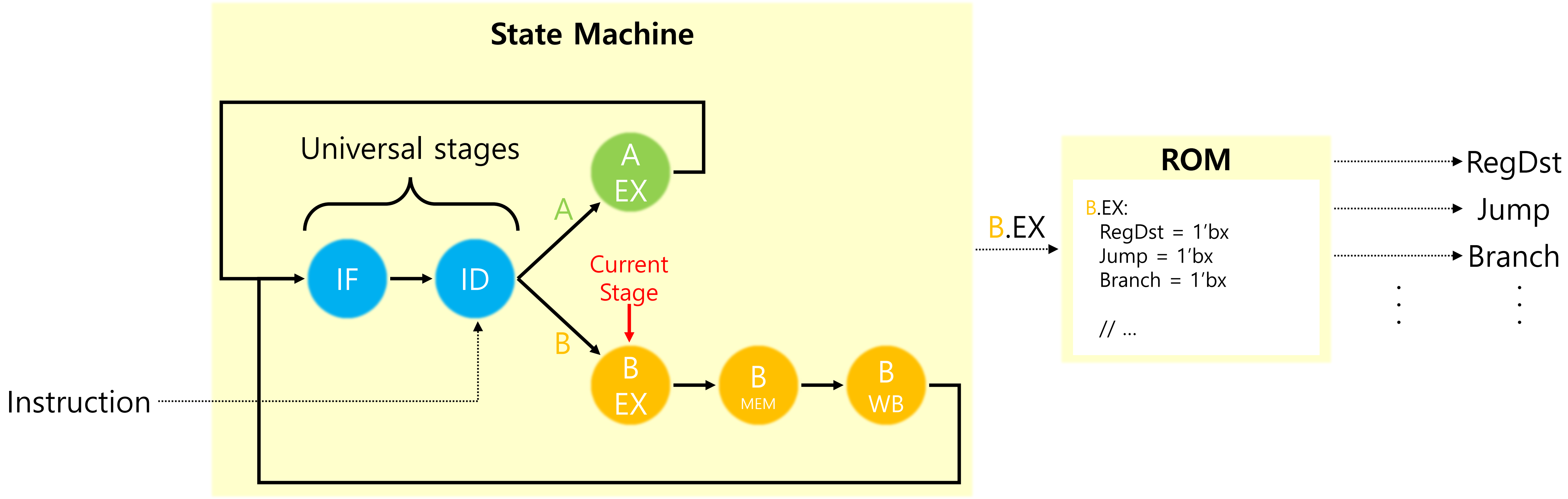
**ROM**

```
B.EX:
  RegDst = 1'bx
  Jump = 1'bx
  Branch = 1'bx

  // ...
```

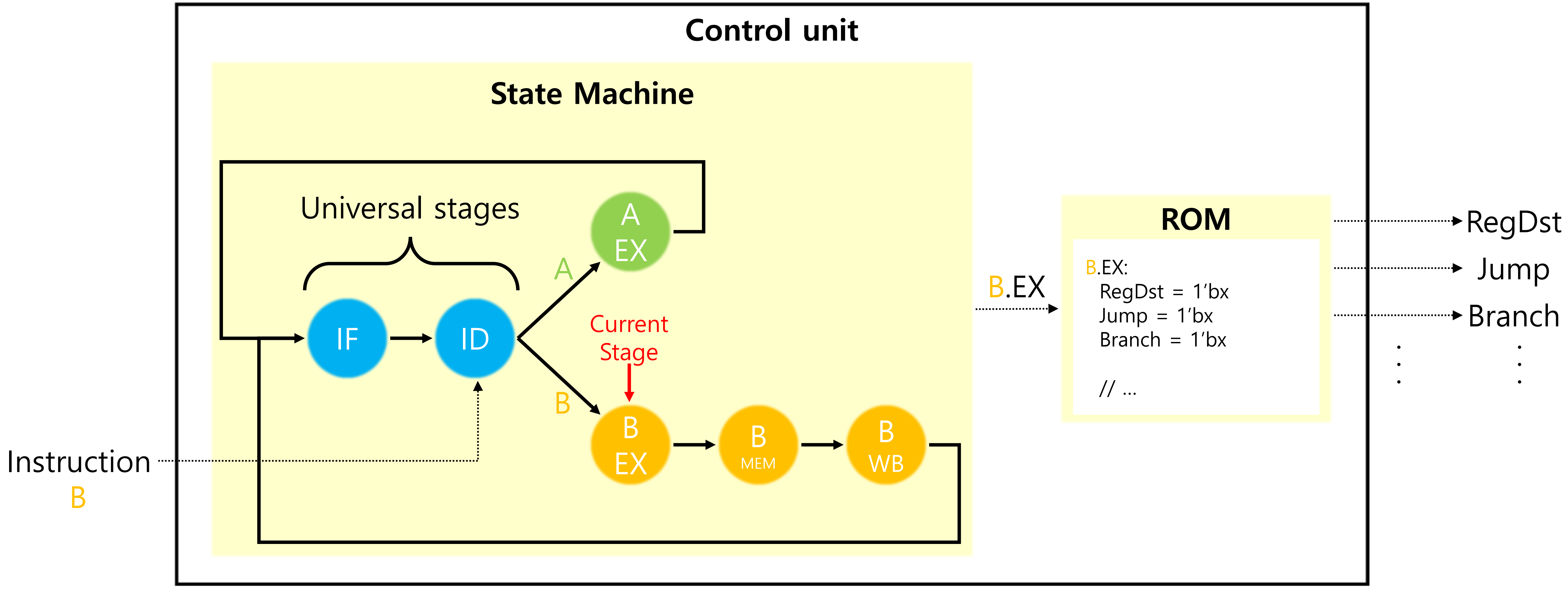
The control signals for each stage are stored in the ROM.

# Control unit of Multi-cycle CPU



The ROM outputs the control signals for the current stage.

# Control unit of Multi-cycle CPU



Put these structures together to build a control unit.

## Datapath of Multi-cycle CPU

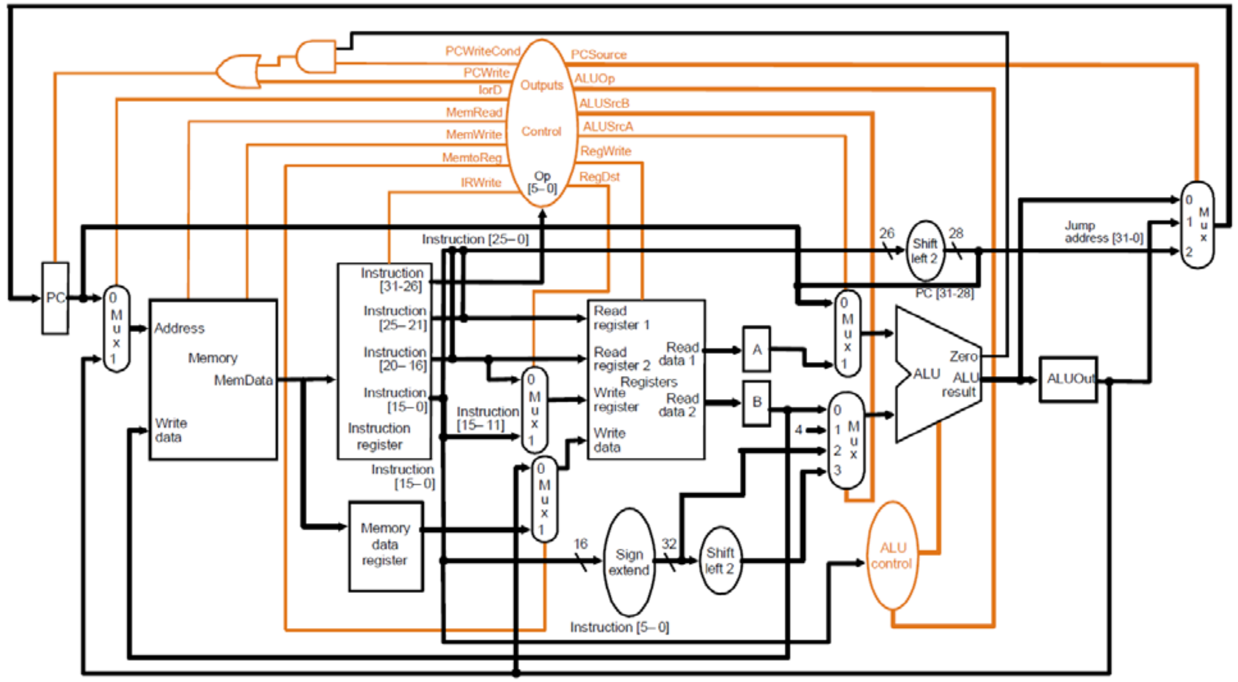
- You may need to revise your ALU:
  - ➢You may want to add some opcodes:
    - ▪ e.g., comparing two integers
  - ➢You may want to add another port:
    - ▪ e.g., an output port for the result of comparison
  - ➢You may want to remove useless opcodes & ports:
    - ▪ e.g., rotate left/right opcodes, Cin/Cout ports

- You can freely revise your previous ALU.

# Datapath of Multi-cycle CPU

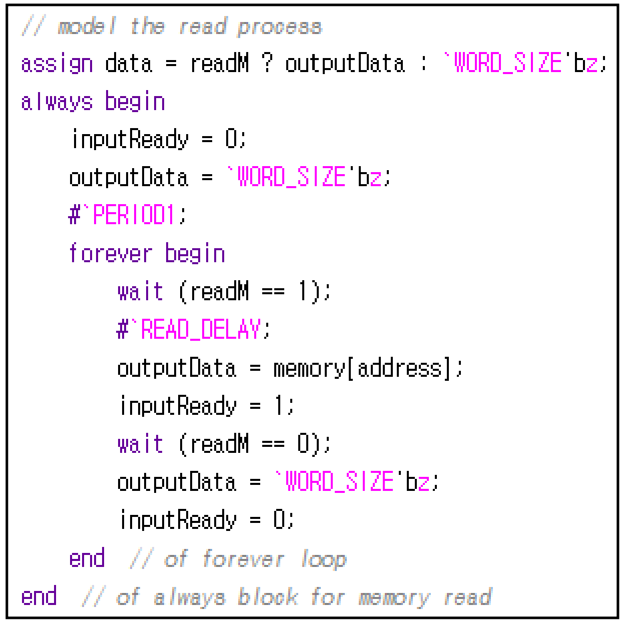
- Your datapath must be capable of executing all TSC instructions.

- The lecture note for the multi-cycle CPU will be the best reference.
    - However, you can do what you want, as long as it works.

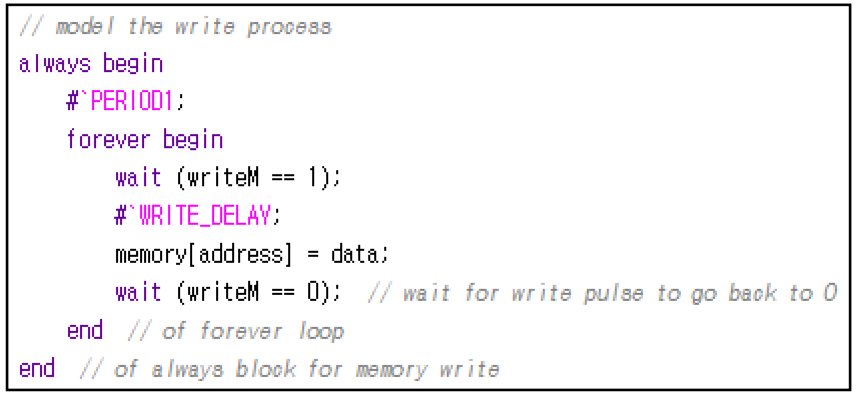


# CPU-Memory interface

- Your CPU has to support Load/Store instructions.
- **PLEASE BE CAUTIOUS**: I revised the memory code.



```
// model the read process
assign data = readM ? outputData : `WORD_SIZE'bz;
always begin
    inputReady = 0;
    outputData = `WORD_SIZE'bz;
    #`PERIOD1;
    forever begin
        wait (readM == 1);
        #`READ_DELAY;
        outputData = memory[address];
        inputReady = 1;
        wait (readM == 0);
        outputData = `WORD_SIZE'bz;
        inputReady = 0;
    end  // of forever loop
end  // of always block for memory read
```

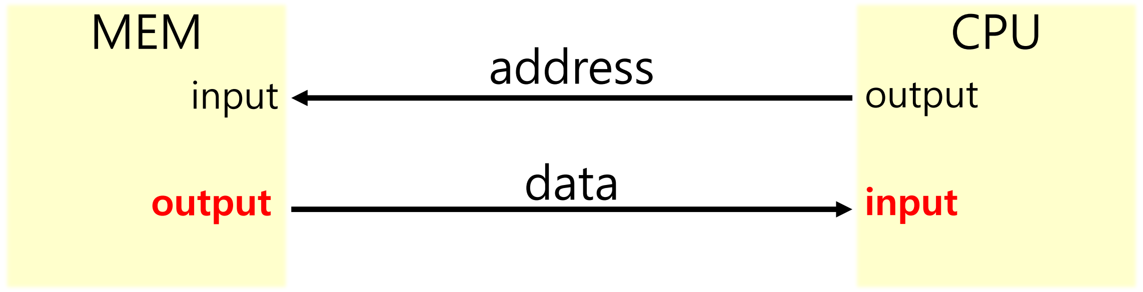


```
// model the write process
always begin
    #`PERIOD1;
    forever begin
        wait (writeM == 1);
        #`WRITE_DELAY;
        memory[address] = data;
        wait (writeM == 0);  // wait for write pulse to go back to 0
    end  // of forever loop
end  // of always block for memory write
```

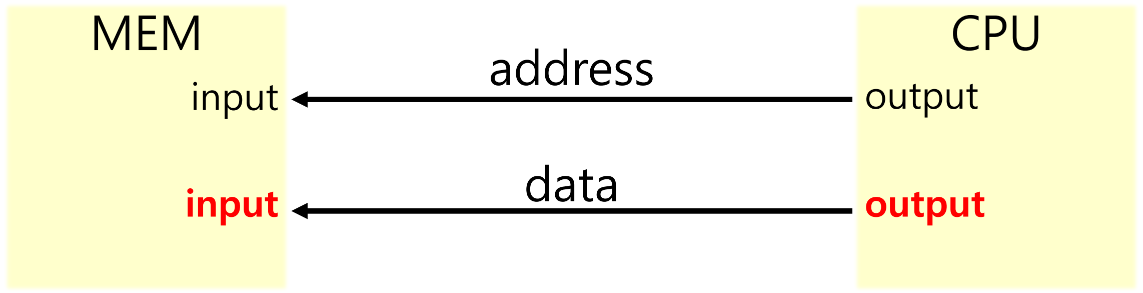
- I removed the “Stable Time”, so *data* keeps its value until you turn the *readM* signal off.

# CPU-Memory interface

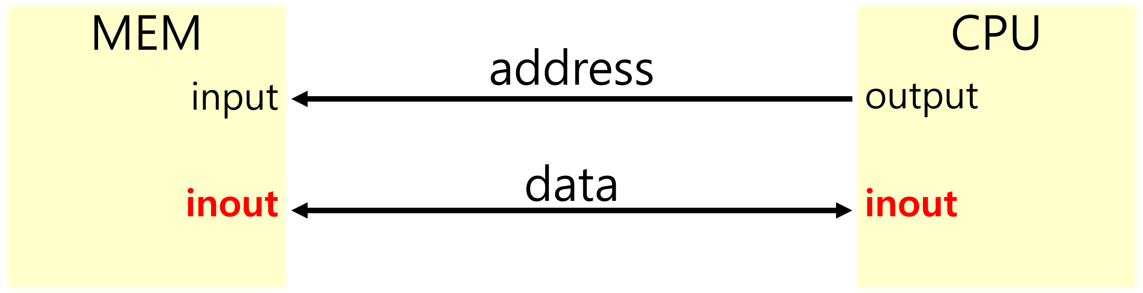
Load-Only



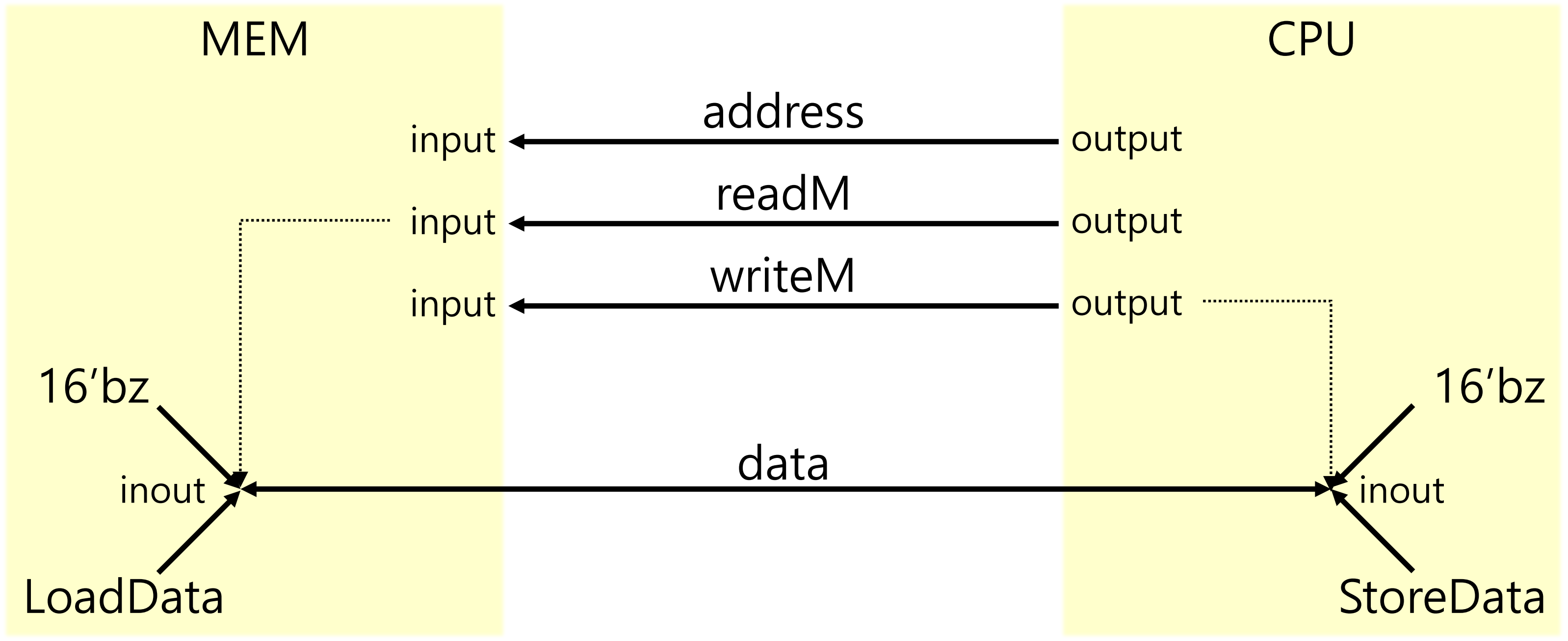
Store-Only



**Load & Store**

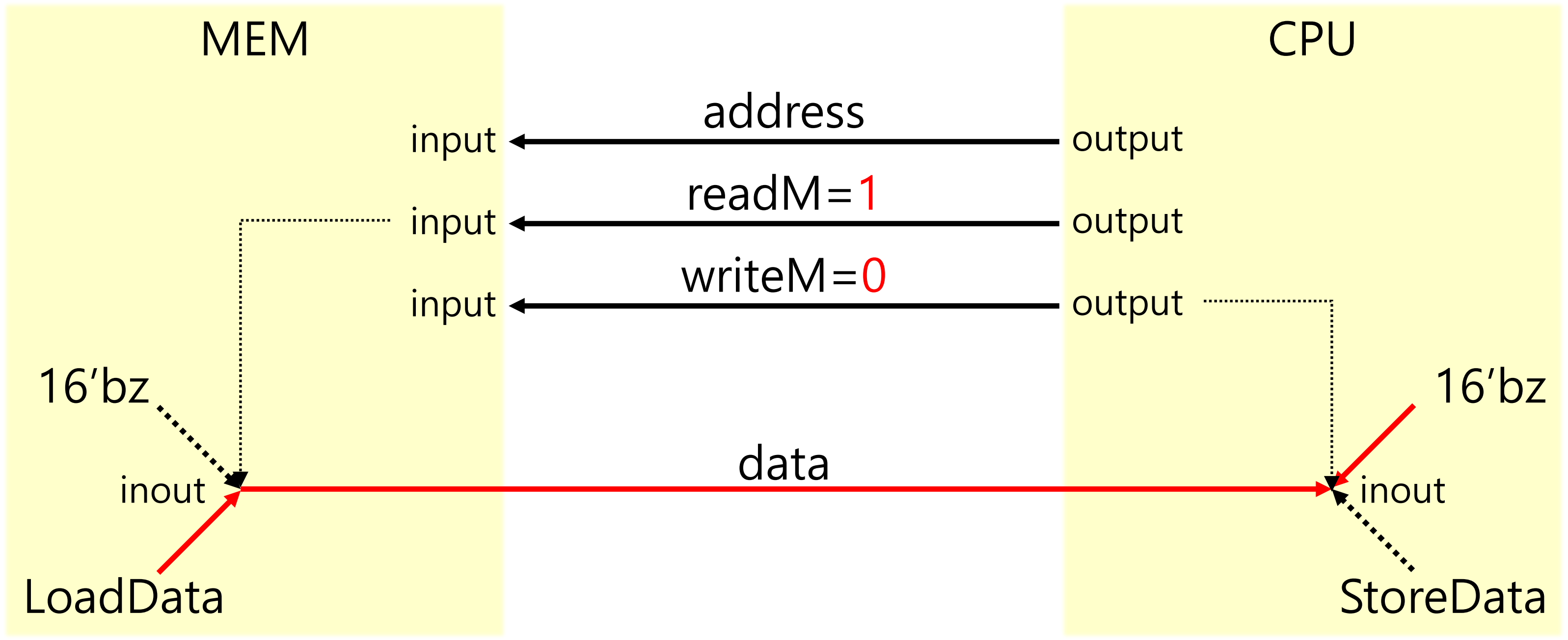


## CPU-Memory interface



'z', a high-impedance state, is a special bit value.

# CPU-Memory interface – Load

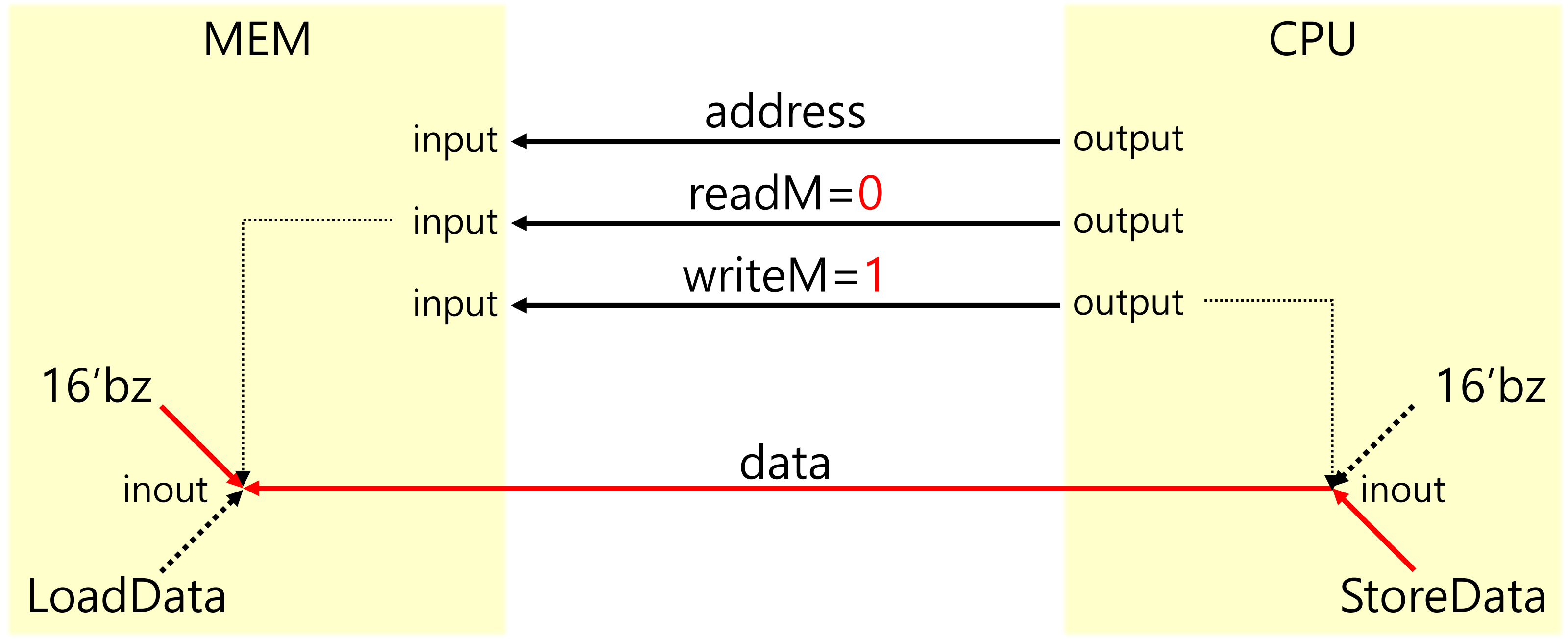


```
assign data = readM ? outputData : `WORD_SIZE'bz;
          outputData = memory[address];
```

(filled by yourself)

The memory puts data on the register, which is assigned to the inout port.

# CPU-Memory interface – Store

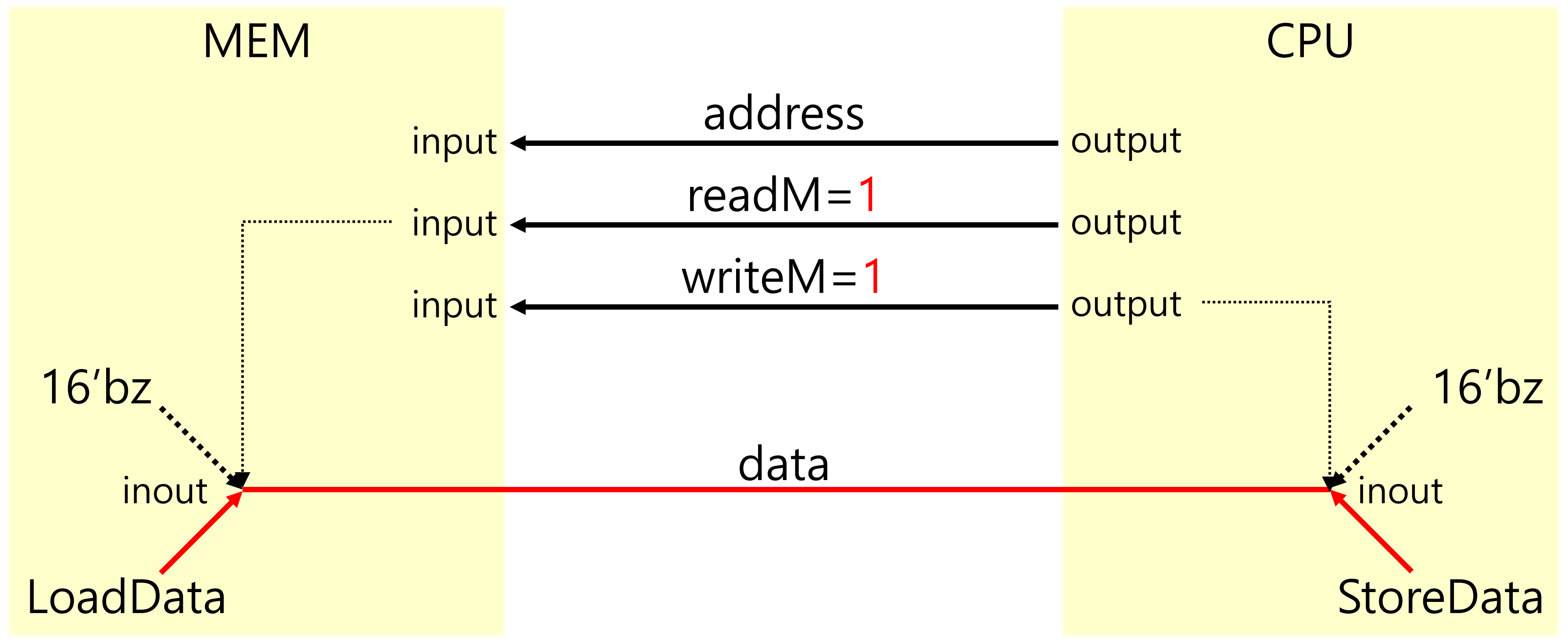


```
assign data = readM ? outputData : `WORD_SIZE'bz;
          memory[address] = data;
```

(filled by yourself)

The memory puts 'z' on the inout port, and reads data from it.

# CPU-Memory interface – Short Circuit



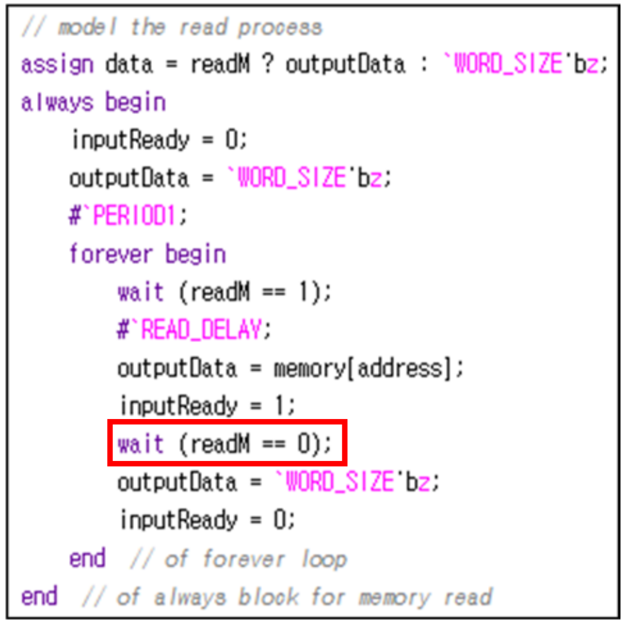
```
assign data = readM ? outputData : `WORD_SIZE'bz;
          outputData = memory[address];
```

(filled by yourself)

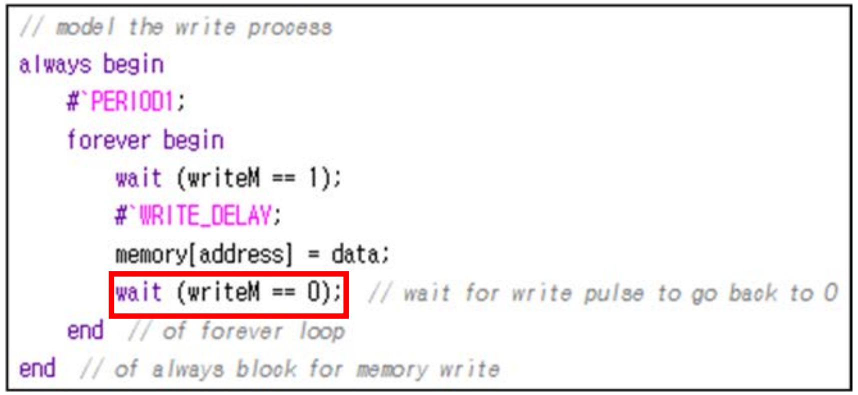
**FORBIDDEN CASE**: Both sides cannot write **SIMULTANEOUSLY.**

# CPU-Memory interface

- **DO NOT FORGET** to turn the readM/writeM off before you use them again.
  - Otherwise, the memory will not respond to you.



```
// model the read process
assign data = readM ? outputData : `WORD_SIZE'bz;
always begin
    inputReady = 0;
    outputData = `WORD_SIZE'bz;
    #`PERIOD1;
    forever begin
        wait (readM == 1);
        #`READ_DELAY;
        outputData = memory[address];
        inputReady = 1;
        wait (readM == 0);
        outputData = `WORD_SIZE'bz;
        inputReady = 0;
    end  // of forever loop
end  // of always block for memory read
```



```
// model the write process
always begin
    #`PERIOD1;
    forever begin
        wait (writeM == 1);
        #`WRITE_DELAY;
        memory[address] = data;
        wait (writeM == 0);  // wait for write pulse to go back to 0
    end  // of forever loop
end  // of always block for memory write
```

# Assignment

- Implement a multi-cycle CPU.
  - It should support full TSC ISA.

- Due date: ~~4/26 23:59:59~~ 4/26 23:59:59
  - -10%/day penalty for late submissions until ~~5/3 23:59:59~~ 5/6 23:59:59
  - 0 point after ~~5/3 23:59:59~~ 5/6 23:59:59

# Evaluation Criteria

- I’ll offer you the testbench consisting of 56 tests.
- This will evaluate the functionality of your CPU

- The first 54 tests (TEST 1-1 to 19-2) validate the functionality of each individual instruction.
- The last two tests (TEST 19-3 and 20) validate the ability of executing streams of instructions.

- The “All Pass!” message shows that your CPU’s functionality is **ALMOST SURELY PERFECT**!

## Evaluation Criteria

- You can freely divide the execution into multiple stages.
  - You can break even “IF” or “ID” into several stages if you want.
- **But, the simpler instruction should have fewer stages.**
  - e.g., The JPR instruction is much simpler than the SWD instruction. Therefore, JPR must have fewer stages than SWD.
- You should remove the redundancies by reusing resources.
  - Please refer to the multicycle lecture note 15~17p
  - **Removing at least one redundancy is mandatory.**

## Testbench Requirements

- You have to supports three output ports for testbench:
    - num_inst : The number of instruction(s) executed
    - output_port : The value written by a WWD instruction
    - is_halted : 1 if and only if the CPU is halted by a HLT instruction.
        - This will stop the simulation and print the results.

- The value of those ports should be valid whenever a positive edge of the clock comes.

Thank You